Arrays:

- An array is a data structure that stores a sequence of values of the same type.
- The data type can be any of Java's *primitive types*:
  - **`int`**, **`short`**, **`byte`**, **`long`**, **`float`**, **`double`**, **`boolean`**, **`char`**
- The data type can also be any *class*:
  - **`String`**, **`SolidBoxes`**, etc.
- Each variable in the array is an element.
- An index specifies the position of each element in the array.
- Useful for many applications:
  - Collecting statistics.
  - Representing the state of a game.
  - Etc.

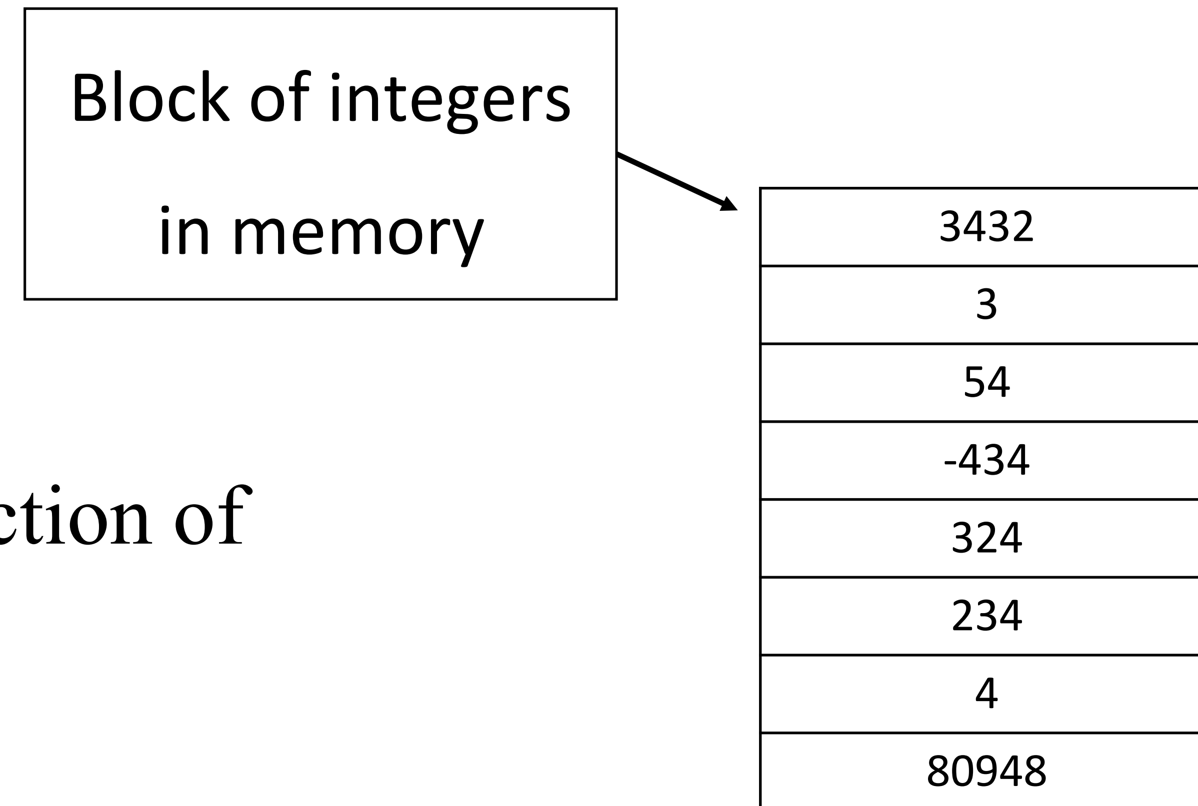
Java Arrays vs Python Lists:

- The closest structure to an array in Python is the List, but there are many differences.
  - An **array** has a fixed size but the size of a **Python List** can change.
  - All the elements of an **array** must be the same type, but a **Python List** can have elements of many types.
  - You may insert into the middle of a **Python List**, but not into an **array**
  - You may concatenate a **Python List**, but not an **array**
- Why would we even have something like an array when a list is so much more flexible?
  - Answer: Nothing is free. Arrays are more efficient than lists.

Arrays

- Arrays are one of the oldest and most basic data structures in computer science.

- Many implementations of arrays use a block of contiguous memory



- It is the most efficient way to store a collection of a known number of items.

- It also allows random access of items. (Through an index)

- In Java, arrays are objects so they contain more information, but the data is stored in consecutive memory.

Declaring and Instantiating Arrays:

- Arrays are objects.
- Creating an array requires two steps:
    1. Declaring the reference to the array
    2. Instantiating the array.
- To declare a reference to the array:

```
datatype []  arrayName;
```

- To instantiate an array:

```
arrayName = new datatype[ size ];
```

  - **size** is an **int** and is the number of elements that will be in the array.

```
int [] zapNumbers;

zapNumbers = new int [ 173 ];

float [] grades;
grades = new float[ 22 ] ;

String [] names;
names = new String[ 20 ];
```

- Examples:
- Declaring and instantiating arrays of *primitive types*:

```
double []  dailyTemps;              // elements are doubles

dailyTemps = new double[ 365 ];  // 365 elements

boolean [] answers;             // elements are booleans: true, false

answers = new boolean[ 20 ];     // 20 elements

int [] cs127A, cs252;          // two arrays, each containing integers

cs127A = new int[ 310 ];         // 310 elements for cs127A

cs252 = new int[ 108 ];           // 108 elements for cs252
```

- Declaring and instantiating arrays of *objects*:

```
String [] cdTracks;          // each element is a String

cdTracks = new String[ 15 ]; // 15 elements to hold song names
BaseballStats[] myTeam;   // BaseballStats is a user defined class
myTeam = new BaseballStats[ 25 ]; // 25 players on my team
```

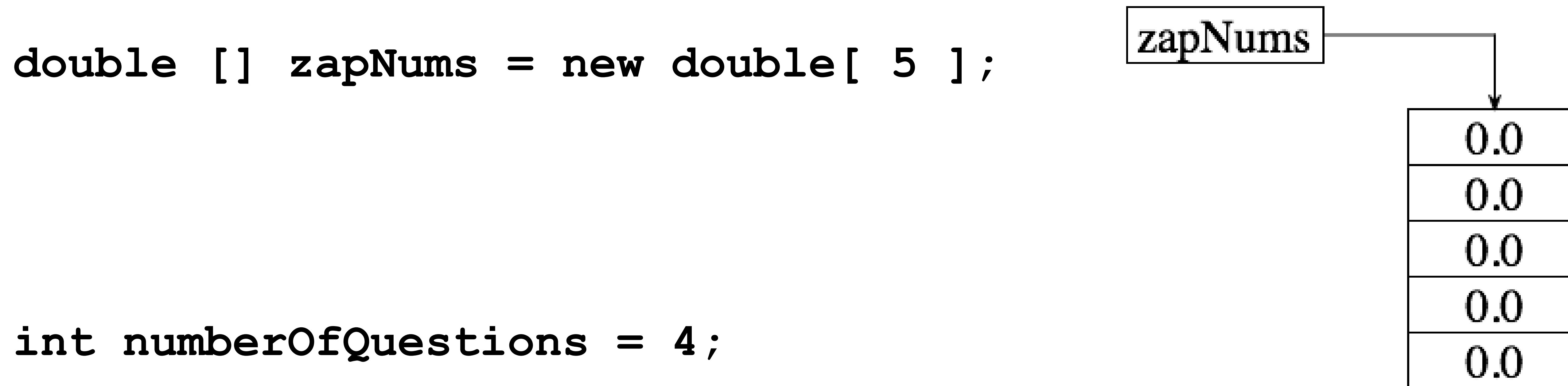
- The declaration and instantiation can be done in the same step.

```
double [] dailyTemps = new double[ 365 ];
... can now use the dailyTemps array in my code ...
dailyTemps = new double[ 173 ];

... can now use the new size of dailyTemps array in my code ...
int numberOfQestions = 30;

boolean [] answers = new boolean[ numberOfQuestions ];

int [] cs127 = new int[ 240 ], cs252 = new int[ 75 ];

String [] studentNames = new String[ 42 ];
int numPlayers = 25;
int numTeams = 10;
BaseballStats [] myTeam = new BaseballStats[ numPlayers * numTeams ];
```

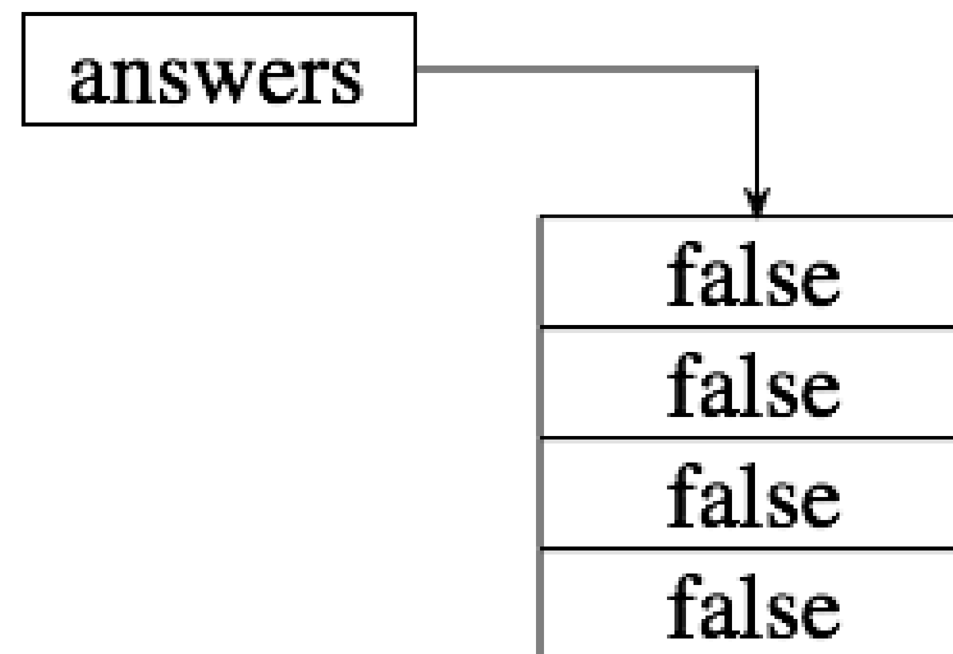
- When an array is instantiated, the elements are assigned default values as follows:

Array data type	Default value
`byte, short, int, long`	**`0`**
`float, double`	**`0`**
`char`	**`nul character`**
`boolean`	**`false`**
Any object reference (for example, a **`String`**)	**`null`**

- The instantiation of an array creates an area of memory that holds the elements of the array.
- This area of memory has one name, the name of the array.




```
double [] zapNums = new double[ 5 ];

int numberOfQuestions = 4;

boolean [] answers = new boolean[ numberOfQuestions ];
```



Assigning initial values to arrays:

- Arrays can be instantiated by specifying a list of initial values.
  - Syntax:

    ```
    datatype [] arrayName = { value0, value1, …};
    ```

    - where **valueN** is an expression evaluating to the data type of the array and is assigned to element at index **N**.

- Examples:
  - Create an array of integers. The array will have **10** elements, since there are **10** values supplied:

```
int magic = 13;
int [] oddNumbers = {1, 3, 5, 7, 9, magic, magic + 2, 17, 19, 21};
System.out.println( oddNumbers[7] );   // prints 17
System.out.println( oddNumbers[4] );   // prints 9
System.out.println( oddNumbers[magic - 4] ); // prints 21
System.out.println( oddNumbers[5] - magic ); // prints 0
```

  - Notes:
    - The **new** keyword is *not* used.
    - The **[ ]**'s are empty; do *not* put in the size of the array.
      - The Java compiler will count the number of elements inside the **{ }**'s and use that as the size of the array.

- Another Example:
  - Create an array of **String**'s. The array will have **3** elements, since there are **3** values supplied:

```
String middleName = "Jane";
String [] myNames = { "Mary", middleName, "Watson" };
```

- You can use this technique only when the array is first declared.
  - For example, the first line below is correct. The second is not!

```
double[] dailyMiles = { 175.3, 278.9, 0.0, 0.0, 0.0}; // correct
dailyMiles = { 170.3, 278.9, 283.2, 158.0, 433.3};    // WRONG!!
```

    - *The compiler will complain about the second line:*

```
zap.java:17: illegal start of expression
     dailyMiles = { 170.3, 278.9, 283.2, 158.0, 433.3};
```

- However, you can use *Anonymous Arrays* to assign values:

```
double[] dailyMiles;
dailyMiles = new double {170.3, 278.9, 283.2, 158.0, 433.3};
```

- The code above works. The last statement is shorthand for:

```
double[] anonymous = {170.3, 278.9, 283.2, 158.0, 433.3};
dailyMiles = anonymous;
```

Accessing Array Elements:

- To access an element of an array, use:

  `arrayName[exp]`

  - where **`exp`** is evaluates to an **`int`** that is >= `0`.
- **`exp`** is the element's *index*; it's position within the array.
- The index of the first element in an array is `0`.
- Each array has a read-only integer instance variable named **`length`**.
  - **`length`** holds the number of elements in the array.
  - Examples:

```
int [] numbers = { 34, 42, 76, 98, -109, 10 };
System.out.println( numbers[0] + " is the element at position 0");
System.out.println("The array numbers has " + numbers.length +
                   " elements");
```

- **Notice the difference between the string *method* length and the array instance *variable* length**

```
String myName = new String("Peter Parker");
System.out.println("My name has " + myName.length() + "characters.");
```

- To access other elements, use either an **`int`**, or an expression that evaluates to an **`int`**:

```
int [] numbers = { 34, 42, 76, 98, -109, 10 };
System.out.println("The element at position 3 is " + numbers[3]);
int sample = 1;
System.out.println("numbers[" + sample + "] is " + numbers[sample]);
System.out.println("numbers[" + sample*2 + "] is " +
                   numbers[sample * 2]);
```

- How to access the last element in the array?
  - The elements in the array are numbered starting at **0**.
  - The **length** instance variable will tell us the number of elements.

```
int lastElement;
lastElement = numbers.length - 1;
System.out.println("last element of numbers is " +
                   numbers[lastElement] );
```

- Trying to access an element at a position < **`0`**, or at a position >= **`arrayName.length`** will generate an error:

```
System.out.println("Element at location -1 is " + numbers[-1]);
```

  - Gives the following error at runtime:

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: -1
        at Sample.main(Sample.java:16)
```

  - Trying to execute:

```
System.out.println("Element at location length is " +
                          numbers[numbers.length]);
```

    - Gives the following error at runtime:

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 6
        at Sample.main(Sample.java:18)
```

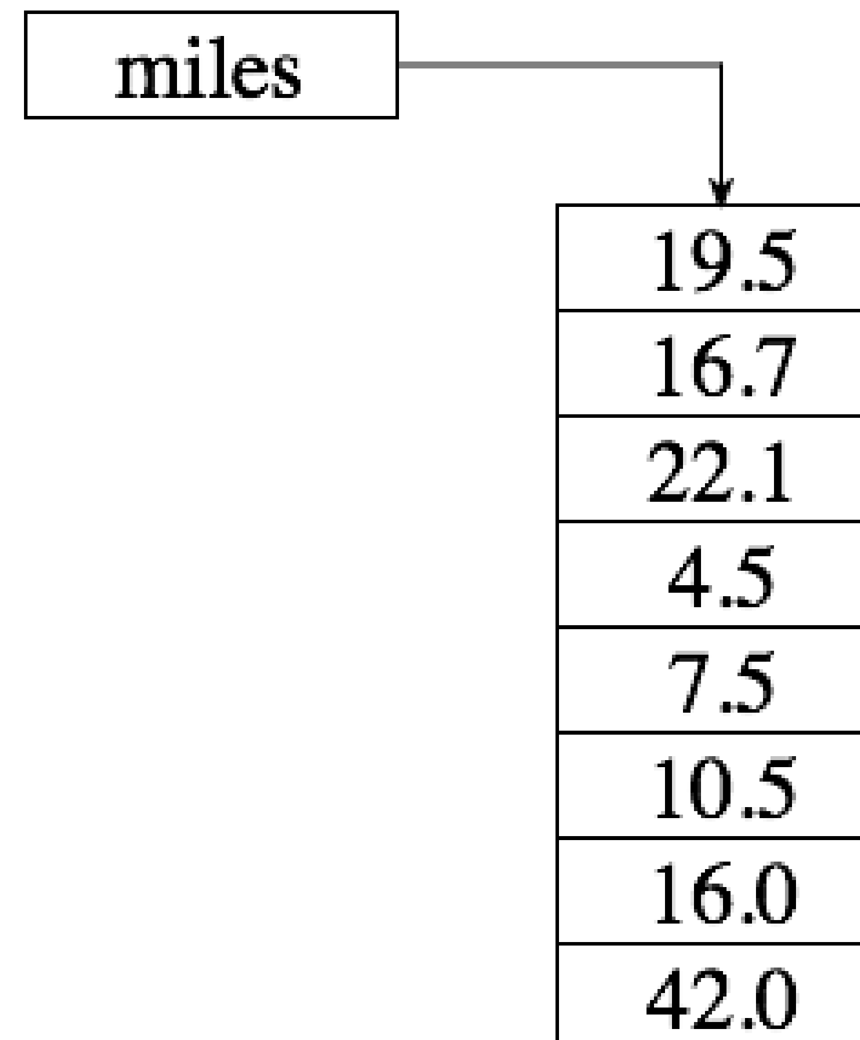
Array Operations:

- Since the elements are indexed starting at **`0`** and going to **`arrayName.length - 1`**, we can print all the elements of an array using a **`for`** loop:

```
public class PrintArray {
   public static void main(String[] args) {
      double [] miles = {19.5, 16.7, 22.1, 4.5, 7.5, 10.5, 16.0, 42.0};
      int i;

      for (i = 0; i < miles.length; i++)
         System.out.println("Element " + i + " is " + miles[i]);

   } // end of method main

} // end of class PrintArray
```

- Gives the following output:

```
Element 0 is 19.5
Element 1 is 16.7
Element 2 is 22.1
Element 3 is 4.5
Element 4 is 7.5
Element 5 is 10.5
Element 6 is 16.0
Element 7 is 42.0
```



Reading data into an array:

- We can read data from the user to put into an array:

```
import java.util.Scanner;
public class ReadDoubles {
   public static void main(String[] args) {
      Scanner inputScan = new Scanner(System.in);
      int i;

      double[] numbers = new double[10];

      for (i = 0; i < numbers.length; i++) {
         System.out.print("Enter a number: ");
         numbers[i] = inputScan.nextDouble();
      }

      System.out.println();

      for (i = 0; i < numbers.length; i++) {
         System.out.print("numbers[" + i + "] is ");
         System.out.println( numbers[i] );
      }
   } // end of method main
} // end of class ReadDoubles
```

Summing elements of an array:

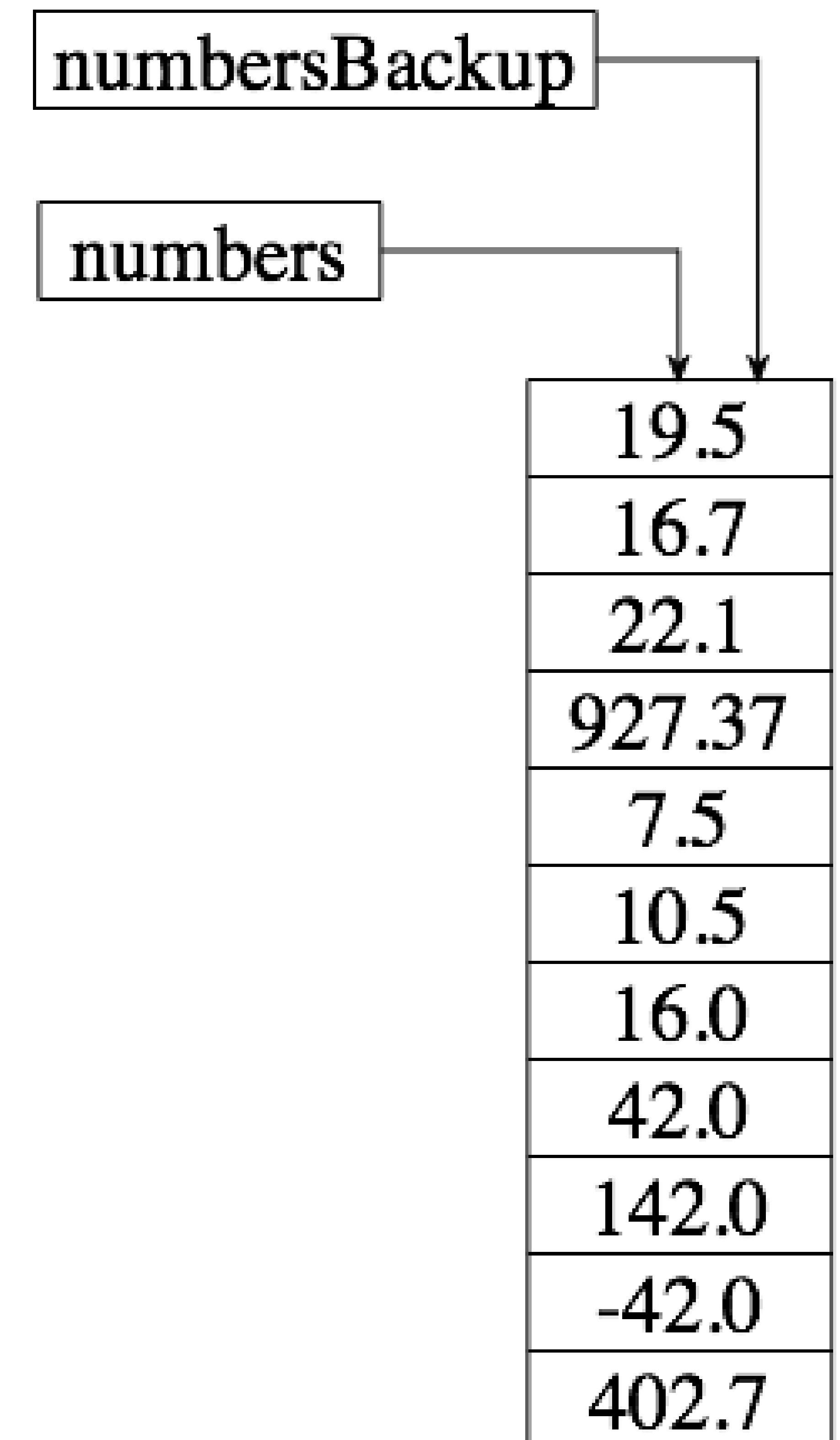
- Once we have the values in an array, we can use loop(s) to perform calculations. For example, we can extend the previous example to find the **average** of the numbers the user enters:

```
double sum;
double average;
double [] numbers = new double[ <some value goes here> ];
... read in the numbers from the user ...

sum = 0.0;
for (i = 0; i < numbers.length; i++) {
   sum = sum + numbers[i];
}

if ( numbers.length != 0 )
   average = sum / numbers.length;
else
   average = 0.0;
System.out.println("average is " + average);
```

Copying arrays:

- Suppose we want to copy the elements of one array to another array. We could try this code:

```
double [] numbersBackup = new double [10];
numbersBackup = numbers;
numbers[3] = 927.37;
System.out.println( numbersBackup[3] );
```



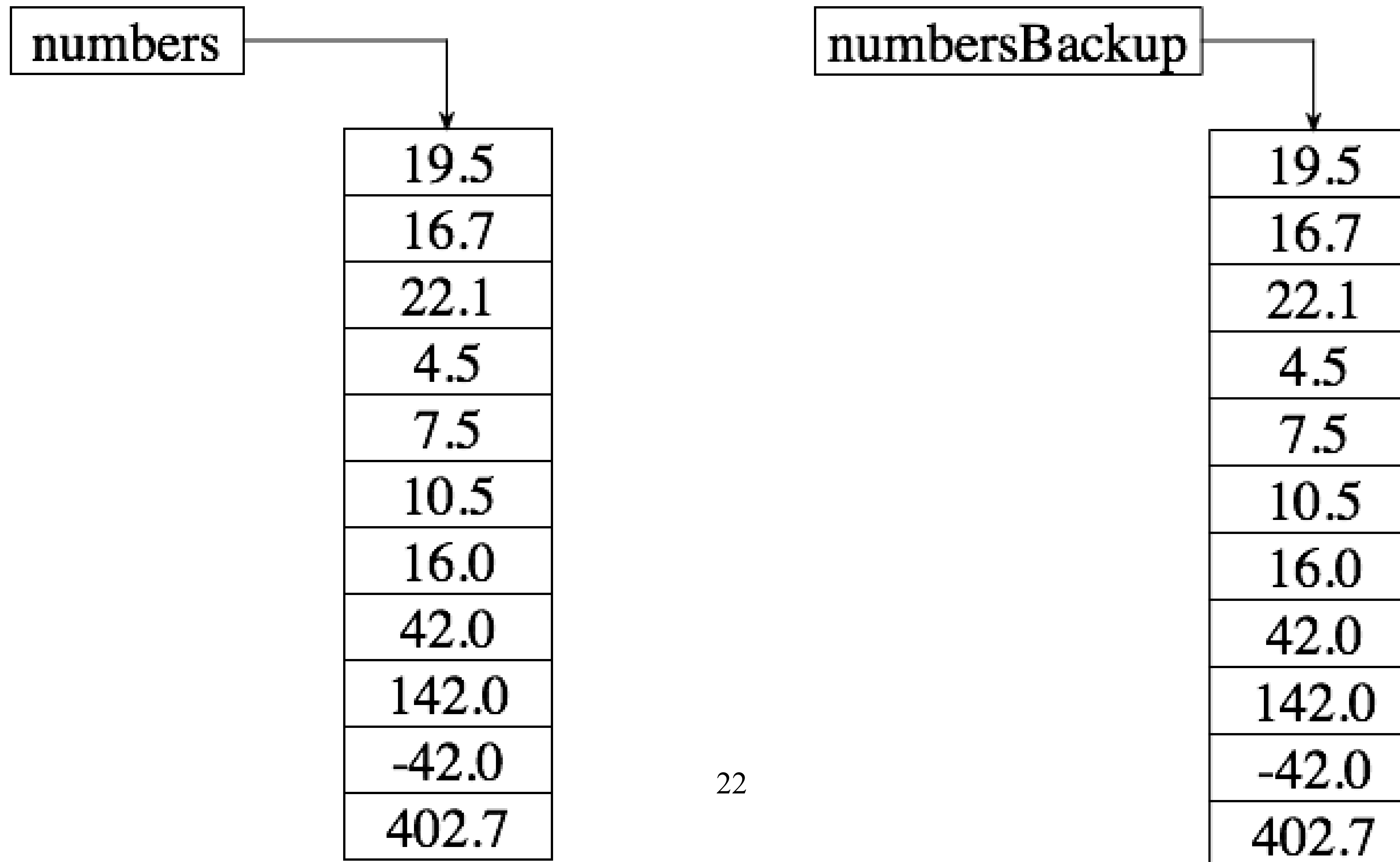
- This code will compile, and it will run without producing error messages.
- But, it is wrong!!!!
- We get two variables that both refer to the same array.
  - Two *aliases* for the same memory location.
- We wanted two arrays with the same contents.

Copying arrays (continued):

- Suppose we want to copy the elements of one array to another array. We could try this code:

```
double [] numbersBackup = new double [ numbers.length ];
for (int i = 0; i < numbers.length; i++)
   numbersBackup[i] = numbers[i];
```

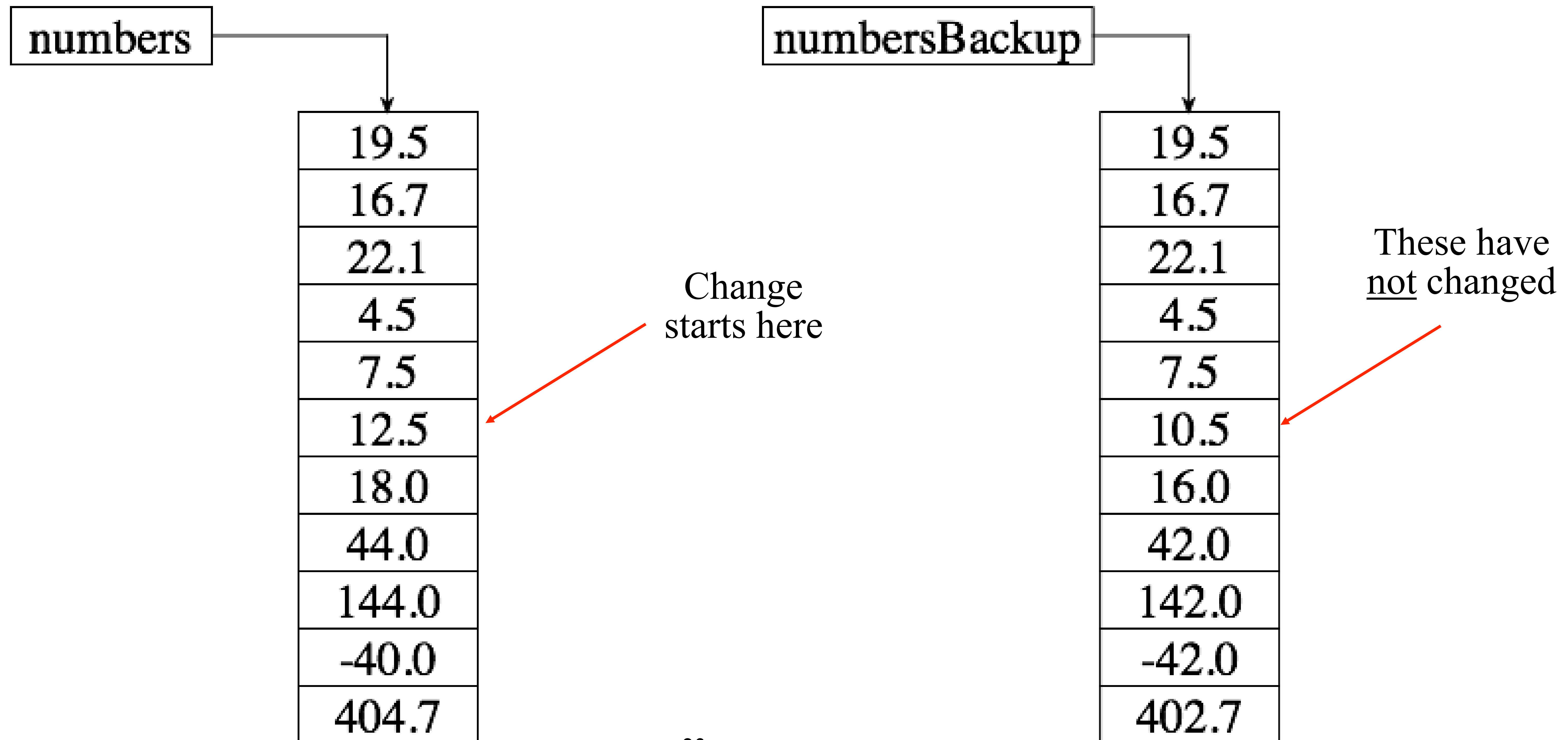
- This code makes a second array and puts a copy of each element from the first array into the second.



Copying arrays (continued):

- Now we can change the contents of one array without changing the other:

```
for (int i = 5; i < numbers.length; i++)
    numbers[i] = numbers[i] + 2;
```



Comparing Arrays for Equality:

```
if ( numbers == numbersBackup )  // does NOT work
```

- To compare whether the elements of two arrays are equal:
  1. Determine if both arrays have the same length.
  2. Compare each element in the first array with the corresponding element in the second array.
- To do this, use a flag variable and a **`for`** loop:
  - A flag variable:
    - A **`boolean`**.
    - Two ways to use it:
      - Set the flag to **`true`**, then test to determine if the flag should be **`false`**. OR
      - Set the flag to **`false`**, then test to determine if the flag should be **`true`**.
  - For this example, we will set the flag to **`true`** and then test to determine if it should be **`false`**.

```
public class CompareArrays {
   public static void main(String[] args) {
      int [] alpha = {0, 1, 2, 43, 48, 59, 60, 70, 88, 90, 1000};
      int [] bravo = {0, 1, 2, 43, 48, 59, 60, 70, 88, 90, 1000};

      boolean equalFlag = true;   // Set the flag to true

      // Test if the flag should be false
      if ( alpha.length != bravo.length )
         equalFlag = false;
      else
         for ( int i = 0; i < alpha.length; i++ ) {
            if ( alpha[i] != bravo[i] )
               equalFlag = false;
               break;
            }

      // Print the result
      if ( equalFlag )
         System.out.println("alpha and bravo are equal");
      else
         System.out.println("alpha and bravo are NOT equal");
   } // end of method main
} // end of class CompareArrays
```

Finding Maximum/Minimum Values:

- General idea:
  - Use a loop to examine each value in the array.
    - Compare each value with the largest value found so far.
- But, what about the first element in the array? What is it compared to?
- Key point: Assume the first value is the largest.
  - Then, the loop tests that assumption, changing the largest as needed.

```
import java.util.Scanner;
public class LargestNumber {
   public static void main(String[] args) {
      Scanner inputScan = new Scanner(System.in);
      int [] zap = new int [10];
      int i;
      int largest;

      for (i = 0; i < zap.length; i++) {
         System.out.print("Enter an integer: ");
         zap[i] = inputScan.nextInt();
      }

      largest = zap[0];   // zap[0] is the largest to start

      // test other values in zap to see if any are larger
      for ( i = 1; i < zap.length; i++ )
         if ( zap[i] > largest )
            largest = zap[i];

      System.out.println("The largest integer is " + largest);
   } // end of method main
} // end of class LargestNumber
```

What would you change to find the smallest instead of the largest?

- What if the array is an array of **String**'s?

```
public class CompareStrings {
   public static void main(String [] args) {
      String [] words = {"hello", "c3p0", "out of here",
                         "outward", "inward", "onward", "r2d2",
                         "water", "Zombie", "wombat"};
      int i;
      String largest;

      largest = words[0];

      for ( i = 1; i < words.length; i++)
         if ( words[i].compareTo(largest) > 0 )
            largest = words[i];

      System.out.println("largest word is " + largest);
   } // end of method main
} // end of class CompareStrings
```

The compareTo method returns a negative number if words[i] < largest, 0 if they are equal (having the same value), and a positive number if words[i] > largest

The Arrays Class:

- The java.util.Arrays class has many useful methods for dealing with arrays, including ones for doing copying and comparing.
- You will want to import java.util.Arrays to use these methods.
- Here is an example of how we could have copied the array in the prior slide.

```
double [] numbersBackup;
numbersBackup = Arrays.copyOf(numbers, number.length);
```

- The general syntax for copyOf is:

```
Arrays.copyOf(<source array>, <length>)
```

- This creates a new array object that has size <length> and whose first element values are copies of those from the source array.
  - If <length> is longer than the length of the source array, the extra elements are given the default values.
  - If <length> is shorter than the length of the source array, only the first elements of that array are copied.

The Arrays Class (cont):

- The **`copyOf`** method is often used to resize an array. This is alright if it is done very infrequently. If you need to resize an array often, you are better of using an **`ArrayList`** object.
- Another useful method in the Arrays class is the **equals** method.
  - Works for arrays of primitive types (int, long, short, char, byte, boolen, float, or double)

    `boolean Arrays.equals(a, b)`

  - Returns true if **a** and **b** have the same length and the elements of in corresponding indexes match and false otherwise.

```
import java.util.*;
public class CompareArrays2 {
   public static void main(String[] args) {
      int [] alpha = {0, 1, 2, 43, 48, 59, 60, 70, 88, 90, 1000};
      int [] bravo = {0, 1, 2, 43, 48, 59, 60, 70, 88, 90, 1000};

      if (Arrays.equals(alpha, bravo))
         System.out.println("alpha and bravo are equal");
      else
         System.out.println("alpha and bravo are NOT equal");
   } // end of method main
} // end of class CompareArrays
```

The Arrays Class (cont):

- Yet another useful method is **sort**.
- The **sort** method works for arrays of primitive types except **boolean**.

void sort(a)

- The sort method sorts the given array using the QuickSort algorithm.
- Notice this method does NOT create a new object, but alters the array given to it as a parameter.

```
import java.util.*;
public class SortArray {
   public static void main(String[] args) {
      int [] alpha = {0, 42, 2, 4, 48, 72, 60, 1000, 90, 88, 70};

      Arrays.sort(alpha);   //
      for (int i = 0; i < alpha.length; ++i)
         System.out.print(alpha[i] + " ");
      System.out.println();
   }
}
```

Command-Line Parameters:

- A java program receives its command-line arguments in an array.

```
public static void main (String[] args) {
```

- Unlike in Python, the first string in the array (index 0) does NOT contain the name of the program, but the first argument sent to it.

```
for (int i = 0; i < args.length; ++i)
  System.out.println(args[i]);
```

  - Suppose main contains the above code and that the program is run by typing:

```
java MyClass John Paul George Ringo
```

  - Then the output will be:

```
John
Paul
George
Ringo
```

The "for each" Loop:

- There is a variation on the for loop that is very similar to that in Python.

```
for (variable : collection) statement
```

- Like in all loops, the statement may actually be a block. The collection may be an array. The variable must be the type of the elements in the collection.
- For example, the loop to print out the arguments shown on the previous slide:

```
for (int i = 0; i < args.length; ++i)
  System.out.println(args[i];
```

- Can be rewritten as:

```
for (String str : args)
  System.out.println(str);
```